FLokk of Kurious Kreatures Reference Manual

0.1

Generated by Doxygen 1.3.8

Fri Dec 10 12:50:01 2004

Contents

Chapter 1

FLokk of Kurious Kreatures Namespace Index

1.1 FLokk of Kurious Kreatures Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

FLokk of Kurious Kreatures Hierarchical Index

2.1 FLokk of Kurious Kreatures Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

FLokk of Kurious Kreatures Class Index

3.1 FLokk of Kurious Kreatures Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

FLokk of Kurious Kreatures File Index

4.1 FLokk of Kurious Kreatures File List

Here is a list of all files with brief descriptions:

Chapter 5

FLokk of Kurious Kreatures Namespace Documentation

5.1

Classes

- class **Agent**
- class **Bound**
- class **Box**
- class **Domain**
- class **Flokk**
- class **LocationalAgent**
- struct **LocationalAgent::Movement**
- class **MotionFlokk**
- class **MotionSolver**
- struct **MotionSolver::Conflict**
- class **Problem**
- class **Sphere**

5.2

Functions

- template<class T> int **planeBoxOverlap** (const Vec< T, 3 > &normal, T d, const Point< T, 3 > &maxbox)
- template<class T> int **triBoxOverlap** (const Point< T, 3 > &boxcenter, const Vec< T, 3 > &boxhalfsize, const Tri< T > &triverts)
- template<class T> bool **intersectLineAAPlane** (unsigned int axis, T offset, const Ray< T > &ray, T &t)
- template<class T> bool **intersect** (const AABox< T > &box, const Tri< T > &tri)
- template<class T> bool **intersect** (const LineSeg< T > &seg, const AABox< T > &box, T &tin, T &tout)
- template<class T> bool **intersect** (const Ray< T > &seg, const AABox< T > &box, T &tin, T &tout)

5.2.1 Function Documentation

5.2.1.1 bool intersect (const Ray< T > & *seg*, const AABox< T > & *box*, T & *tin*, T & *tout*)

Definition at line 169 of file IntersectionExt.h.

References intersectLineAAPlane().

```
170    {
171       float t;
172       tin = -1;
173       tout = 10000;
174
175
176       //if parallel to the box in any way, check bounds
177       if ( seg.mDir[0] == 0 )
178       {
179          if ( seg.mOrigin[0]>box.mMax[0] || seg.mOrigin[0]<box.mMin[0] ) return false;
180       }
181       else if ( seg.mDir[1] == 0 )
182       {
183          if ( seg.mOrigin[1]>box.mMax[1] || seg.mOrigin[1]<box.mMin[1] ) return false;
184       }
185       else if ( seg.mDir[2] == 0 )
186       {
187          if ( seg.mOrigin[2]>box.mMax[2] || seg.mOrigin[2]<box.mMin[2] ) return false;
188       }
189       //test each plane of the box, and if an intersection occurs, cut away at tin and tout, until the [tin,tout] s
190       //is fully bound by the box, or tout crosses tin, in which case it was a miss.
191       //Source: "Computer Graphics using OpenGL, Second Edition" pg. 754
192
193
194       if ( intersectLineAAPlane( 0,box.mMin[0],seg,t ) )
195       {
196          if ( seg.mDir[0]>0 ) tin = t > tin ? t : tin;
197          else           tout = t < tout ? t : tout;
198
199          if ( tin>tout )return false;
200
201       }
202
203       if ( intersectLineAAPlane( 0,box.mMax[0],seg,t ) )
204       {
```

```
205          if ( seg.mDir[0]<0 ) tin = t > tin ? t : tin;
206          else           tout = t < tout ? t : tout;
207
208          if ( tin>tout )return false;
209
210       }
211
212       if ( intersectLineAAPlane( 1,box.mMin[1],seg,t ) )
213       {
214          if ( seg.mDir[1]>0 ) tin = t > tin ? t : tin;
215          else           tout = t < tout ? t : tout;
216
217          if ( tin>tout )return false;
218
219       }
220
221       if ( intersectLineAAPlane( 1,box.mMax[1],seg,t ) )
222       {
223          if ( seg.mDir[1]<0 ) tin = t > tin ? t : tin;
224          else           tout = t < tout ? t : tout;
225
226          if ( tin>tout )return false;
227
228       }
229
230       if ( intersectLineAAPlane( 2,box.mMin[2],seg,t ) )
231       {
232          if ( seg.mDir[2]>0 ) tin = t > tin ? t : tin;
233          else           tout = t < tout ? t : tout;
234
235          if ( tin>tout )return false;
236
237       }
238
239       if ( intersectLineAAPlane( 2,box.mMax[2],seg,t ) )
240       {
241          if ( seg.mDir[2]<0 ) tin = t > tin ? t : tin;
242          else           tout = t < tout ? t : tout;
243
244          if ( tin>tout )return false;
245
246       }
247
248       if ( tin>tout )return false;
249       return true;
250
251    }
```

5.2.1.2 bool intersect (const LineSeg< T > & *seg*, const AABox< T > & *box*, T & *tin*, T & *tout*)

given a line segment and an axis aligned bounding box, returns whether the line intersects the box, and if so, tin and tout are set to the parametric terms on the line segment where the segment enters and exits the box respectively Definition at line 81 of file IntersectionExt.h.

References intersectLineAAPlane().

```
82    {
83       float t;
84       tin = -1;
85       tout = 10000;
86
87
88       //if parallel to the box in any way, check bounds
```

```
89       if ( seg.mDir[0] == 0 )
90       {
91          if ( seg.mOrigin[0]>box.mMax[0] || seg.mOrigin[0]<box.mMin[0] ) return false;
92       }
93       else if ( seg.mDir[1] == 0 )
94       {
95          if ( seg.mOrigin[1]>box.mMax[1] || seg.mOrigin[1]<box.mMin[1] ) return false;
96       }
97       else if ( seg.mDir[2] == 0 )
98       {
99          if ( seg.mOrigin[2]>box.mMax[2] || seg.mOrigin[2]<box.mMin[2] ) return false;
100       }
101       //test each plane of the box, and if an intersection occurs, cut away at tin and tout, until the [tin,tout] s
102       //is fully bound by the box, or tout crosses tin, in which case it was a miss.
103       //Source: "Computer Graphics using OpenGL, Second Edition" pg. 754
104
105
106       if ( intersectLineAAPlane( 0,box.mMin[0],seg,t ) )
107       {
108          if ( seg.mDir[0]>0 ) tin = t > tin ? t : tin;
109          else           tout = t < tout ? t : tout;
110
111          if ( tin>tout )return false;
112
113       }
114
115       if ( intersectLineAAPlane( 0,box.mMax[0],seg,t ) )
116       {
117          if ( seg.mDir[0]<0 ) tin = t > tin ? t : tin;
118          else           tout = t < tout ? t : tout;
119
120          if ( tin>tout )return false;
121
122       }
123
124       if ( intersectLineAAPlane( 1,box.mMin[1],seg,t ) )
125       {
126          if ( seg.mDir[1]>0 ) tin = t > tin ? t : tin;
127          else           tout = t < tout ? t : tout;
128
129          if ( tin>tout )return false;
130
131       }
132
133       if ( intersectLineAAPlane( 1,box.mMax[1],seg,t ) )
134       {
135          if ( seg.mDir[1]<0 ) tin = t > tin ? t : tin;
136          else           tout = t < tout ? t : tout;
137
138          if ( tin>tout )return false;
139
140       }
141
142       if ( intersectLineAAPlane( 2,box.mMin[2],seg,t ) )
143       {
144          if ( seg.mDir[2]>0 ) tin = t > tin ? t : tin;
145          else           tout = t < tout ? t : tout;
146
147          if ( tin>tout )return false;
148
149       }
150
151       if ( intersectLineAAPlane( 2,box.mMax[2],seg,t ) )
152       {
153          if ( seg.mDir[2]<0 ) tin = t > tin ? t : tin;
154          else           tout = t < tout ? t : tout;
155
```

```
156          if ( tin>tout )return false;
157
158       }
159
160       if ( tin>tout )return false;
161       tout = tout > 1 ? 1 : tout;
162       if ( tin>1 )return false;
163       return true;
164
165    }
```

5.2.1.3 bool intersect (const AABox< T > & *box*, const Tri< T > & *tri*)

given an axis aligned bounding box and a triangle, returns whether the triangle intersects the box anywhere Definition at line 69 of file IntersectionExt.h.

Referenced by Sphere::contains(), Box::contains(), and Box::intersect().

```
70    {
71       Point<T,3> center = (box.mMax + box.mMin)/2;
72       Vec<T,3> half = box.mMax - center;
73       return gmtl::triBoxOverlap<T>(center, half, tri );
74    }
```

5.2.1.4 bool intersectLineAAPlane (unsigned int *axis*, T *offset*, const Ray< T > & *ray*, T & *t*)

given an axis of 0 - 2 (for x y z) and an offset on the specified axis, calculates the intersection between the line (as defined by a ray along it) and an axis aligned plane setting t to be the parametric term on the line where it intersects the plane Definition at line 56 of file IntersectionExt.h.

Referenced by intersect().

```
57    {
58       gmtlASSERT( axis<3 && "axis out of bounds in boolIntersectRayAAPlane(...)" );
59       if ( ray.mDir[axis] == 0 ) return false;
60       t = ( offset - ray.mOrigin[axis]) / (ray.mDir[axis]);
61       return true;
62    }
```

5.2.1.5 int planeBoxOverlap (const Vec< T, 3 > & *normal*, T *d*, const Point< T, 3 > & *maxbox*)

Definition at line 284 of file IntersectionExt.h.

References DOT, X, and Z.

```
285    {
286       int q;
287       T vmin[3],vmax[3];
288       for ( q=X;q<=Z;q++ )
289       {
290          if ( normal[q]>0.0f )
291          {
292             vmin[q]=-maxbox[q];
```

```
293             vmax[q]=maxbox[q];
294          }
295          else
296          {
297             vmin[q]=maxbox[q];
298             vmax[q]=-maxbox[q];
299          }
300       }
301       if ( DOT(normal,vmin)+d>0.0f ) return 0;
302       if ( DOT(normal,vmax)+d>=0.0f ) return 1;
303
304       return 0;
305    }
```

5.2.1.6 int triBoxOverlap (const Point< T, 3 > & *boxcenter*, const Vec< T, 3 > & *boxhalfsize*, const Tri< T > & *triverts*)

Definition at line 358 of file IntersectionExt.h.

References AXISTEST_X01, AXISTEST_X2, AXISTEST_Y02, AXISTEST_Y1, AXISTEST_Z0, AXISTEST_Z12, FINDMINMAX, SUB, X, Y, and Z.

```
359    {
360
361       /*    use separating axis theorem to test overlap between triangle and box */
362       /*    need to test for overlap in these directions: */
363       /*    1) the {x,y,z}-directions (actually, since we use the AABB of the triangle */
364       /*       we do not even need to test these) */
365       /*    2) normal of the triangle */
366       /*    3) crossproduct(edge from tri, {x,y,z}-directin) */
367       /*       this gives 3x3=9 more tests */
368       Vec<T,3> v0,v1,v2;
369       Vec<T,3> axis;
370       T min,max,d,p0,p1,p2,rad,fex,fey,fez;
371       Vec<T,3> normal,e0,e1,e2;
372
373       /* This is the fastest branch on Sun */
374       /* move everything so that the boxcenter is in (0,0,0) */
375       SUB(v0,triverts[0],boxcenter);
376       SUB(v1,triverts[1],boxcenter);
377       SUB(v2,triverts[2],boxcenter);
378
379       /* compute triangle edges */
380       SUB(e0,v1,v0);      /* tri edge 0 */
381       SUB(e1,v2,v1);      /* tri edge 1 */
382       SUB(e2,v0,v2);      /* tri edge 2 */
383
384       /* Bullet 3:  */
385       /*  test the 9 tests first (this was faster) */
386       fex = fabs(e0[X]);
387       fey = fabs(e0[Y]);
388       fez = fabs(e0[Z]);
389       AXISTEST_X01(e0[Z], e0[Y], fez, fey);
390       AXISTEST_Y02(e0[Z], e0[X], fez, fex);
391       AXISTEST_Z12(e0[Y], e0[X], fey, fex);
392
393       fex = fabs(e1[X]);
394       fey = fabs(e1[Y]);
395       fez = fabs(e1[Z]);
396       AXISTEST_X01(e1[Z], e1[Y], fez, fey);
397       AXISTEST_Y02(e1[Z], e1[X], fez, fex);
398       AXISTEST_Z0(e1[Y], e1[X], fey, fex);
399
400       fex = fabs(e2[X]);
```

```
401       fey = fabs(e2[Y]);
402       fez = fabs(e2[Z]);
403       AXISTEST_X2(e2[Z], e2[Y], fez, fey);
404       AXISTEST_Y1(e2[Z], e2[X], fez, fex);
405       AXISTEST_Z12(e2[Y], e2[X], fey, fex);
406
407       /* Bullet 1: */
408       /*  first test overlap in the {x,y,z}-directions */
409       /*  find min, max of the triangle each direction, and test for overlap in */
410       /*  that direction -- this is equivalent to testing a minimal AABB around */
411       /*  the triangle against the AABB */
412
413       /* test in X-direction */
414       FINDMINMAX(v0[X],v1[X],v2[X],min,max);
415       if ( min>boxhalfsize[X] || max<-boxhalfsize[X] ) return 0;
416
417       /* test in Y-direction */
418       FINDMINMAX(v0[Y],v1[Y],v2[Y],min,max);
419       if ( min>boxhalfsize[Y] || max<-boxhalfsize[Y] ) return 0;
420
421       /* test in Z-direction */
422       FINDMINMAX(v0[Z],v1[Z],v2[Z],min,max);
423       if ( min>boxhalfsize[Z] || max<-boxhalfsize[Z] ) return 0;
424
425       /* Bullet 2: */
426       /*  test if the box intersects the plane of the triangle */
427       /*  compute plane equation of triangle: normal*x+d=0 */
428       gmtl::cross(normal,e0,e1);
429       d=-gmtl::dot(normal,v0);  /* plane eq: normal.x+d=0 */
430       if ( !planeBoxOverlap<T>(normal,d,boxhalfsize) ) return 0;
431
432       return 1;   /* box and triangle overlaps */
433    }
```

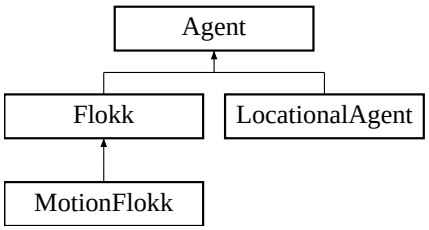
Chapter 6

FLokk of Kurious Kreatures Class Documentation

6.1 Agent Class Reference

`#include <Agent.h>`

Inheritance diagram for Agent::



Public Member Functions

- **Agent** ()
- virtual ~**Agent** ()
- void **setDefaultProblem** (**Problem** *new_default)
- void **addProblem** (**Problem** *p)
- **Flokk** * **getParent** () const
- void **setParent** (**Flokk** *parent)
- virtual void **think** (float dt)
- void **setUserData** (void *data)
- const void * **getUserData** () const
- void * **getUserData** ()

Protected Attributes

- **Flokk** * **mParent**
- **Problem** * **mDefaultProblem**

- std::vector< **Problem** * > **mProblems**
- void * **mUserData**

6.1.1 Constructor & Destructor Documentation

6.1.1.1 Agent ()

Definition at line 37 of file Agent.cpp.

References Agent::mDefaultProblem, and Agent::mParent.

```
38    {
39       mDefaultProblem = NULL;
40       mParent = NULL;
41    }
```

6.1.1.2 ~Agent () [virtual]

Definition at line 43 of file Agent.cpp.

```
44    {
45    }
```

6.1.2 Member Function Documentation

6.1.2.1 void addProblem (Problem * *p*)

Used to add a **Problem**(p. 52) to the list of Problems that need solving. Definition at line 82 of file Agent.cpp.

References Agent::mProblems, and Problem::setAgent().

```
83    {
84       if ( p )
85       {
86          mProblems.push_back(p);
87          p->setAgent( this );
88       }
89    }
```

6.1.2.2 Flokk * getParent () const

Returns the **Flokk**(p. 31) this agent belongs to. Null if the agent is a root **Flokk**(p. 31) Definition at line 48 of file Agent.cpp.

References Agent::mParent.

```
49    {
50       return mParent;
51    }
```

6.1.2.3 void* getUserData () [inline]

Definition at line 88 of file Agent.h.

References Agent::mUserData.

```
89          {
90             return mUserData;
91          }
```

6.1.2.4 const void* getUserData () const [inline]

Definition at line 83 of file Agent.h.

References Agent::mUserData.

```
84          {
85             return mUserData;
86          }
```

6.1.2.5 void setDefaultProblem (Problem * *new_default*)

Used to set the defaul **Problem**(p. 52) that will be used when no other problems are available to solve. Definition at line 91 of file Agent.cpp.

References Agent::mDefaultProblem, and Problem::setAgent().

```
92    {
93       if ( mDefaultProblem != NULL )
94       {
95          delete mDefaultProblem;
96       }
97       mDefaultProblem = new_default;
98       mDefaultProblem->setAgent( this );
99    }
```

6.1.2.6 void setParent (Flokk * *parent*)

Definition at line 53 of file Agent.cpp.

References Agent::mParent.

Referenced by Flokk::addAgent().

```
54    {
55       //if( mParent != NULL ) remove ourselves from parent?
56       mParent = parent;
57    }
```

6.1.2.7 void setUserData (void * *data*) [inline]

Definition at line 78 of file Agent.h.

References Agent::mUserData.

```
79          {
80             mUserData = data;
81          }
```

6.1.2.8 void think (float *dt*) [virtual]

Function that programmer fills in with any **Agent**(p. 17) specific updating that needs to happen that doesn't concern moving such as refreshing a list of specific goals that it is trying to accomplish.

Reimplemented in **LocationalAgent** (p. 38), and **MotionFlokk** (p. 43).

Definition at line 59 of file Agent.cpp.

References Problem::isSolved(), Agent::mDefaultProblem, Agent::mProblems, Problem::reset(), and Problem::solve().

```
60    {
61       if ( mProblems.empty() )
62       {
63          mDefaultProblem->solve( dt );
64
65          if ( mDefaultProblem->isSolved() )
66          {
67             mDefaultProblem->reset();
68          }
69       }
70       else
71       {
72          mProblems[0]->solve( dt );
73
74          if ( mProblems[0]->isSolved() )
75          {
76             delete mProblems[0];
77             mProblems.erase(mProblems.begin());
78          }
79       }
80    }
```

6.1.3 Member Data Documentation

6.1.3.1 Problem* mDefaultProblem [protected]

Definition at line 96 of file Agent.h.

Referenced by Agent::Agent(), Agent::setDefaultProblem(), and Agent::think().

6.1.3.2 Flokk* mParent [protected]

Definition at line 95 of file Agent.h.

Referenced by Agent::Agent(), Agent::getParent(), and Agent::setParent().

6.1.3.3 std::vector<Problem*> mProblems [protected]

Definition at line 97 of file Agent.h.

Referenced by Agent::addProblem(), and Agent::think().

6.1.3.4 void* mUserData [protected]

Definition at line 98 of file Agent.h.

Referenced by Agent::getUserData(), and Agent::setUserData().

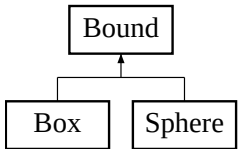
The documentation for this class was generated from the following files:

- **Agent.h**
- **Agent.cpp**

6.2 Bound Class Reference

`#include <Bound.h>`

Inheritance diagram for Bound::



Public Member Functions

- void **setTransform** (const gmtl::Matrix44f &m)
- const gmtl::Matrix44f & **getTransform** () const
- const gmtl::Matrix44f & **getInvTransform** () const
- void **setPosition** (float x, float y, float z)
- gmtl::Point3f **getPosition** () const
- virtual bool **contains** (const gmtl::Point3f &p)=0
- virtual bool **intersect** (const gmtl::LineSegf &line, int &numHits, float &hit1, float &hit2)=0
- virtual gmtl::AABoxf **getBoundingBox** () const =0

6.2.1 Detailed Description

Bound(p. 22) The base class for the types of shapes that can define a domain.

Definition at line 44 of file Bound.h.

6.2.2 Member Function Documentation

6.2.2.1 virtual bool contains (const gmtl::Point3f & *p*) [pure virtual]

Does this bound contain the point p?

Implemented in **Box** (p. 25), and **Sphere** (p. 54).

6.2.2.2 virtual gmtl::AABoxf getBoundingBox () const [pure virtual]

Returns the axis-aligned bounding box of this bound

Implemented in **Box** (p. 26), and **Sphere** (p. 55).

6.2.2.3 const gmtl::Matrix44f & getInvTransform () const

Get the cached inverse of the transformation matrix. Definition at line 48 of file Bound.cpp.

Referenced by Box::contains(), Sphere::intersect(), and Box::intersect().

```
49    {
50       return mInvTransform;
51    }
```

6.2.2.4 gmtl::Point3f getPosition () const

Get the center of the bound in space Definition at line 63 of file Bound.cpp.

Referenced by Sphere::contains().

```
64    {
65       return gmtl::Point3f( mTransform(0,3), mTransform(1,3), mTransform(2,3) );
66    }
```

6.2.2.5 const gmtl::Matrix44f & getTransform () const

Get the full transformation matrix of the bound. Definition at line 43 of file Bound.cpp.

Referenced by Box::getBoundingBox().

```
44    {
45       return mTransform;
46    }
```

6.2.2.6 virtual bool intersect (const gmtl::LineSegf & *line*, int & *numHits*, float & *hit1*, float & *hit2*) [pure virtual]

Determines if a given line segment intersects this bound, how many times it intersects, and where along the line segment parametrically, indicated by hit1 and hit2 which both range from 0 - 1.

Implemented in **Box** (p. 27), and **Sphere** (p. 55).

6.2.2.7 void setPosition (float *x*, float *y*, float *z*)

Set the center of the bound in space Definition at line 53 of file Bound.cpp.

References Bound::setTransform().

```
54    {
55       mTransform(0,3) = x;
56       mTransform(1,3) = y;
57       mTransform(2,3) = z;
58
59       //invoke the inverse caching at the end of setTransform
60       setTransform( mTransform );
61    }
```

6.2.2.8 void setTransform (const gmtl::Matrix44f & *m*)

Set the transformation matrix that defines where in space this bound is located. Definition at line 37 of file Bound.cpp.

Referenced by Bound::setPosition().

```
38    {
39       mTransform = m;
40       mInvTransform = m;
41       gmtl::invert( mInvTransform );
42    }
```

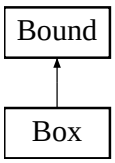
The documentation for this class was generated from the following files:

- **Bound.h**
- **Bound.cpp**

6.3 Box Class Reference

`#include <Box.h>`

Inheritance diagram for Box::



Public Member Functions

- **Box** ()
- **Box** (float xSize, float ySize, float zSize)
- void **setSize** (float xSize, float ySize, float zSize)
- const gmtl::Point3f & **getMin** () const
- const gmtl::Point3f & **getMax** () const
- virtual bool **contains** (const gmtl::Point3f &p)
- virtual bool **intersect** (const gmtl::LineSegf &line, int &numHits, float &hit1, float &hit2)
- virtual gmtl::AABoxf **getBoundingBox** () const

6.3.1 Constructor & Destructor Documentation

6.3.1.1 Box ()

Create a unit box at the origin (of dimensions 1,1,1). Definition at line 38 of file Box.cpp.

```
39    {
40       mBox.setMin( gmtl::Point3f(-0.5f,-0.5f,-0.5f) );
41       mBox.setMax( gmtl::Point3f( 0.5f, 0.5f, 0.5f) );
42    }
```

6.3.1.2 Box (float *xSize*, float *ySize*, float *zSize*)

Create a box of dimensions 'xSize','ySize', 'zSize' at the origin. Definition at line 44 of file Box.cpp.

```
45    {
46       mBox.setMin( gmtl::Point3f(-0.5f * xSize,-0.5f * ySize,-0.5f * zSize) );
47       mBox.setMax( gmtl::Point3f( 0.5f * xSize, 0.5f * ySize, 0.5f * zSize) );
48    }
```

6.3.2 Member Function Documentation

6.3.2.1 bool contains (const gmtl::Point3f & *p*) [virtual]

Does this bound contain the point p?

Implements **Bound** (p. 22).

Definition at line 66 of file Box.cpp.

References Bound::getInvTransform(), and gmtl::intersect().

```
67    {
68       //p transformed into local space
69       gmtl::Point3f pt = getInvTransform() * p;
70       return gmtl::intersect( mBox, pt );
71    }
```

6.3.2.2 virtual gmtl::AABoxf getBoundingBox () const [inline, virtual]

Returns the axis-aligned bounding box of this bound

Implements **Bound** (p. 22).

Definition at line 77 of file Box.h.

References Bound::getTransform().

```
78       {
79          gmtl::Point3f vertices[8];
80          gmtl::Point3f mn = mBox.getMin();
81          gmtl::Point3f mx = mBox.getMax();
82
83          vertices[0] = ( getTransform() * mn );
84          vertices[1] = ( getTransform() * mx );
85
86          vertices[2] = ( getTransform() * gmtl::Point3f( mn[0], mn[1], mx[2] ) );
87          vertices[3] = ( getTransform() * gmtl::Point3f( mx[0], mn[1], mn[2] ) );
88          vertices[4] = ( getTransform() * gmtl::Point3f( mx[0], mn[1], mx[2] ) );
89
90          vertices[5] = ( getTransform() * gmtl::Point3f( mn[0], mx[1], mx[2] ) );
91          vertices[6] = ( getTransform() * gmtl::Point3f( mx[0], mx[1], mn[2] ) );
92          vertices[7] = ( getTransform() * gmtl::Point3f( mx[0], mx[1], mx[2] ) );
93
94          for ( int v = 0; v<8; ++v )
95          {
96             for ( int i = 0; i<3; ++i )
97             {
98                mn[i] = mn[i] < vertices[v][i] ? mn[i] : vertices[v][i];
99             }
100         }
101
102         for ( int v = 0; v<8; ++v )
103         {
104            for ( int i = 0; i<3; ++i )
105            {
106               mx[i] = mx[i] > vertices[v][i] ? mx[i] : vertices[v][i];
107            }
108         }
109         return gmtl::AABoxf( mn, mx );
110      }
```

6.3.2.3 const gmtl::Point3f & getMax () const

Returns the corner of the box with the largest local coordinates Definition at line 61 of file Box.cpp.

```
62    {
63       return mBox.getMax();
64    }
```

6.3.2.4 const gmtl::Point3f & getMin () const

Returns the corner of the box with the smallest local coordinates Definition at line 56 of file Box.cpp.

```
57    {
58       return mBox.getMin();
59    }
```

6.3.2.5 bool intersect (const gmtl::LineSegf & *line*, int & *numHits*, float & *hit1*, float & *hit2*) [virtual]

Determines if a given line segment intersects this bound, how many times it intersects, and where along the line segment parametrically, indicated by hit1 and hit2 which both range from 0 - 1.

Implements **Bound** (p. 23).

Definition at line 73 of file Box.cpp.

References Bound::getInvTransform(), and gmtl::intersect().

```
74    {
75       gmtl::LineSegf lt = line;
76       lt.mOrigin = getInvTransform() * lt.mOrigin;
77       lt.mDir = getInvTransform() * lt.mDir;
78
79       //TODO: if this doesn't work, the intersection function doesn't handle points inside the box
80
81       numHits = 0;
82       bool result = gmtl::intersect( lt, mBox, hit1, hit2 );
83       if ( result ) numHits = 2;
84       if ( gmtl::intersect( mBox, lt.mOrigin ) || gmtl::intersect( mBox, gmtl::Point3f( lt.mOrigin + lt.mDir )) )
85       {
86          numHits = 1;
87       }
88       return result;
89    }
```

6.3.2.6 void setSize (float *xSize*, float *ySize*, float *zSize*)

Defines the dimensions of the box along the local x, y, and z axis. The box will be centered around it's dimensions. Definition at line 50 of file Box.cpp.

```
51    {
52       mBox.setMin( gmtl::Point3f(-0.5f * xSize,-0.5f * ySize,-0.5f * zSize) );
53       mBox.setMax( gmtl::Point3f( 0.5f * xSize, 0.5f * ySize, 0.5f * zSize) );
54    }
```

The documentation for this class was generated from the following files:

- **Box.h**
- **Box.cpp**

6.4 Domain Class Reference

`#include <Domain.h>`

Public Member Functions

- **Domain** ()
- void **addNegativeBound** (**Bound** *b)
- void **addPositiveBound** (**Bound** *b)
- bool **inDomain** (const gmtl::Point3f &p)
- gmtl::AABoxf **getBoundingBox** () const
- bool **clipToDomain** (gmtl::LineSegf &result, const gmtl::LineSegf &line)

6.4.1 Detailed Description

Domain(p. 28) This class defines the area agents are allowed to wander in. This includes physical bounds as well as obstacles constructed out of GMTL primitives.

Definition at line 49 of file Domain.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Domain ()

Definition at line 37 of file Domain.cpp.

```
38    {
39    }
```

6.4.3 Member Function Documentation

6.4.3.1 void addNegativeBound (Bound * b)

Add a bound agents should stay out of. Definition at line 41 of file Domain.cpp.

```
42    {
43       if ( b ) mNegativeBounds.push_back( b );
44    }
```

6.4.3.2 void addPositiveBound (Bound * b)

Add a bound agents should stay inside of. Definition at line 46 of file Domain.cpp.

```
47    {
48       if ( b ) mPositiveBounds.push_back( b );
49    }
```

6.4.3.3 bool clipToDomain (gmtl::LineSegf & *result*, const gmtl::LineSegf & *line*) [inline]

Shortens the end of the line segment to make it fit inside the domain and write's the resulting line segment to 'result'. The function returns whether the line was clipped or not. Definition at line 87 of file Domain.h.

References Domain::inDomain().

```
88       {
89          result = line;
90          //TODO: this is an error really
91          if ( !inDomain( line.getOrigin() ) ) return false;
92
93          if ( mNegativeBounds.empty() && mPositiveBounds.empty() ) return false;
94
95          //nearest parametric value of intersection
96          float nearest = 1.0f;
97
98          float hit1, hit2;
99          int numHits;
100          for ( size_t i = 0; i<mNegativeBounds.size(); ++i )
101          {
102             if ( mNegativeBounds[i]->intersect( line, numHits, hit1, hit2 ) )
103             {
104                if ( hit1 < nearest )
105                {
106                   nearest = hit1;
107                }
108                if ( numHits > 1 && hit2 < nearest )
109                {
110                   nearest = hit2;
111                }
112             }
113          }
114
115          //TODO: this is a hack, figure out how to do all positive bounds
116          if ( !mPositiveBounds.empty() )
117          {
118             int numHits;
119             float hit1;
120             float hit2;
121             if ( mPositiveBounds[0]->intersect( line, numHits, hit1, hit2 ) )
122             {
123                float farthest = hit1;
124                if ( numHits > 1 && hit2 > hit1 ) farthest = hit2;
125                if ( farthest < nearest )
126                {
127                   nearest = farthest;
128                }
129             }
130          }
131
132          if ( nearest < 1.0f )
133          {
134             result.mDir *= nearest;
135             return true;
136          }
137
138          /*
139          //find all intersections, sort them by distance, then do sweep for exit from domain ()-style
140            for( size_t i = 0; i<mPositiveBounds.size(); ++i )
141            {
142               if( mPositiveBounds[i]->intersect( line, thit ) && thit < nearest )
143            {
144               nearest = thit;
```

```
145            }
146            }*/
147
148          return false;
149       }
```

6.4.3.4 gmtl::AABoxf getBoundingBox () const [inline]

Definition at line 74 of file Domain.h.

Referenced by MotionSolver::setDomain().

```
75       {
76          if ( mPositiveBounds.empty() ) return gmtl::AABoxf( gmtl::Point3f(-1000,-1000,-1000),gmtl::Point3f(1000,1000
77          //TODO: make this work for more than one positive bound
78          return mPositiveBounds[0]->getBoundingBox();
79       }
```

6.4.3.5 bool inDomain (const gmtl::Point3f & *p*)

Is the given point within positive and outside of negative bounds? Definition at line 51 of file Domain.cpp.

Referenced by Domain::clipToDomain().

```
52    {
53       //TODO: this is (0)=2*n*m, an oct tree or datastructure for the bounds
54       //would be cool.
55       for ( size_t i = 0; i<mNegativeBounds.size(); ++i )
56       {
57          if ( mNegativeBounds[i]->contains( p ) ) return false;
58       }
59
60       for ( size_t i = 0; i<mPositiveBounds.size(); ++i )
61       {
62          if ( !(mPositiveBounds[i]->contains( p )) ) return false;
63       }
64       return true;
65    }
```

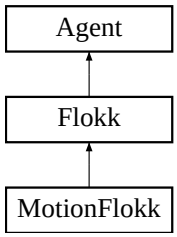
The documentation for this class was generated from the following files:

- **Domain.h**
- **Domain.cpp**

6.5 Flokk Class Reference

`#include <Flokk.h>`

Inheritance diagram for Flokk::



Public Member Functions

- **Flokk** ()
- virtual ~**Flokk** ()
- virtual void **addAgent** (**Agent** *a)
- virtual void **requestProblem** (**Problem** *p)

Protected Attributes

- std::vector< **Agent** * > **mAgents**
- std::vector< **Problem** * > **mRequests**

6.5.1 Constructor & Destructor Documentation

6.5.1.1 Flokk ()

Definition at line 37 of file Flokk.cpp.

```
37                  : Agent()
38    {
39    }
```

6.5.1.2 ~Flokk () [virtual]

Definition at line 41 of file Flokk.cpp.

```
42    {
43    }
```

6.5.2 Member Function Documentation

6.5.2.1 void addAgent (Agent * *a*) [virtual]

Registers an agent with this flokk. Definition at line 45 of file Flokk.cpp.

References Flokk::mAgents, and Agent::setParent().

```
46    {
47       if ( !a ) return;
48       mAgents.push_back( a );
49       a->setParent( this );
50    }
```

6.5.2.2 void requestProblem (Problem * *p*) [virtual]

Definition at line 52 of file Flokk.cpp.

References Flokk::mRequests.

```
53    {
54       if ( !p ) return;
55       //TODO: do more than this?
56       mRequests.push_back( p );
57    }
```

6.5.3 Member Data Documentation

6.5.3.1 std::vector<Agent*> mAgents [protected]

Definition at line 58 of file Flokk.h.

Referenced by Flokk::addAgent().

6.5.3.2 std::vector<Problem*> mRequests [protected]

Definition at line 59 of file Flokk.h.

Referenced by Flokk::requestProblem().

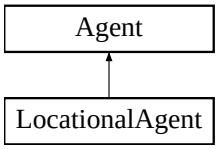
The documentation for this class was generated from the following files:

- **Flokk.h**
- **Flokk.cpp**

6.6 LocationalAgent Class Reference

`#include <LocationalAgent.h>`

Inheritance diagram for LocationalAgent::



Public Member Functions

- **LocationalAgent** ()
- virtual ~**LocationalAgent** ()
- const gmtl::AABoxf & **getAABox** () const
- void **setAABox** (const gmtl::AABoxf &aabox)
- const gmtl::Point3f & **getPosition** () const
- const gmtl::Point3f & **getDesiredPosition** () const
- void **setDesiredPosition** (const gmtl::Point3f &desiredPos)
- void **setMaxSpeed** (float s)
- float **getMaxSpeed** () const
- float **getSpeed** () const
- float **getDesiredSpeed** () const
- void **setDesiredSpeed** (float speed)
- const gmtl::Matrix33f & **getOrientation** () const
- const gmtl::Matrix33f & **getDesiredOrientation** () const
- void **setDesiredOrientation** (const gmtl::Matrix33f &desiredRot)
- virtual void **think** (float dt)
- void **setPosition** (const gmtl::Point3f &p)
- void **setDirection** (const gmtl::Vec3f &v)
- const gmtl::Vec3f & **getDirection** () const

Protected Member Functions

- void **setSpeed** (float speed)
- void **setOrientation** (const gmtl::Matrix33f &rot)

Protected Attributes

- gmtl::Point3f **mPosition**
- gmtl::Point3f **mDesiredPosition**
- gmtl::Vec3f **mDirection**
- float **mSpeed**
- float **mDesiredSpeed**
- float **mMaxSpeed**
- gmtl::Matrix33f **mOrientation**
- gmtl::Matrix33f **mDesiredOrientation**
- gmtl::AABoxf **mAABox**

6.6.1 Constructor & Destructor Documentation

6.6.1.1 LocationalAgent ()

Definition at line 39 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredSpeed, LocationalAgent::mDirection, LocationalAgent::mMaxSpeed, and LocationalAgent::mSpeed.

```
39                                      : Agent()
40    {
41       mSpeed = 0;
42       mDesiredSpeed = 0;
43       mMaxSpeed = -1;
44           mDirection.set( 0,0,1 );
45    }
```

6.6.1.2 ~LocationalAgent () [virtual]

Definition at line 47 of file LocationalAgent.cpp.

```
48    {
49    }
```

6.6.2 Member Function Documentation

6.6.2.1 const gmtl::AABoxf & getAABox () const

Definition at line 56 of file LocationalAgent.cpp.

References LocationalAgent::mAABox.

```
57    {
58       return mAABox;
59    }
```

6.6.2.2 const gmtl::Matrix33f & getDesiredOrientation () const

Returns the Agent's current desired orientation as a rotation matrix. Definition at line 91 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredOrientation.

```
92    {
93       return mDesiredOrientation;
94    }
```

6.6.2.3 const gmtl::Point3f & getDesiredPosition () const

Returns the Agent's current desired position. Definition at line 71 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredPosition.

Referenced by MotionFlokk::think().

```
72    {
73       return mDesiredPosition;
74    }
```

6.6.2.4 float getDesiredSpeed () const

Returns the Agent's desired speed. Definition at line 131 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredSpeed.

Referenced by MotionFlokk::think().

```
132    {
133       return mDesiredSpeed;
134    }
```

6.6.2.5 const gmtl::Vec3f & getDirection () const

Returns the agent's current direction vectorDefinition at line 142 of file LocationalAgent.cpp.

References LocationalAgent::mDirection.

```
143    {
144       return mDirection;
145    }
```

6.6.2.6 float getMaxSpeed () const

Returns the Agent's maximum speed.Definition at line 126 of file LocationalAgent.cpp.

References LocationalAgent::mMaxSpeed.

Referenced by MotionFlokk::think().

```
127    {
128       return mMaxSpeed;
129    }
```

6.6.2.7 const gmtl::Matrix33f & getOrientation () const

Returns the Agent's current orientation as a rotation matrix. Definition at line 86 of file Locational-Agent.cpp.

References LocationalAgent::mOrientation.

```
87    {
88       return mOrientation;
89    }
```

6.6.2.8 const gmtl::Point3f & getPosition () const

Returns the Agent's current position. Definition at line 66 of file LocationalAgent.cpp.

References LocationalAgent::mPosition.

Referenced by MotionFlokk::think().

```
67    {
68       return mPosition;
69    }
```

6.6.2.9 float getSpeed () const

Returns the Agent's current speed. Definition at line 121 of file LocationalAgent.cpp.

References LocationalAgent::mSpeed.

Referenced by MotionFlokk::think().

```
122    {
123       return mSpeed;
124    }
```

6.6.2.10 void setAABox (const gmtl::AABoxf & *aabox*)

Definition at line 61 of file LocationalAgent.cpp.

References LocationalAgent::mAABox.

```
62    {
63       mAABox = aabox;
64    }
```

6.6.2.11 void setDesiredOrientation (const gmtl::Matrix33f & *desiredRot*)

Sets the Agent's desired orientation to the rotation matrix passed in. Definition at line 101 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredOrientation.

```
102    {
103       mDesiredOrientation = desiredOrient;
104    }
```

6.6.2.12 void setDesiredPosition (const gmtl::Point3f & *desiredPos*)

Sets the Agent's desired position to the value passed in. Definition at line 81 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredPosition.

```
82    {
83       mDesiredPosition = desiredPos;
84    }
```

6.6.2.13 void setDesiredSpeed (float *speed*)

Sets the Agent's desired speed to the value passed in. Definition at line 116 of file LocationalAgent.cpp.

References LocationalAgent::mDesiredSpeed.

```
117    {
118       mDesiredSpeed = s;
119    }
```

6.6.2.14 void setDirection (const gmtl::Vec3f & *v*)

Sets the agent's current direction to the value passed in. Should only be used initially for simulation setup. Definition at line 136 of file LocationalAgent.cpp.

References LocationalAgent::mDirection.

```
137    {
138       mDirection = v;
139           gmtl::normalize( mDirection );
140    }
```

6.6.2.15 void setMaxSpeed (float *s*)

Sets the Agent's maximum speed. Definition at line 111 of file LocationalAgent.cpp.

References LocationalAgent::mMaxSpeed.

```
112    {
113       mMaxSpeed = s;
114    }
```

6.6.2.16 void setOrientation (const gmtl::Matrix33f & *rot*) [protected]

Sets the Agent's current orientation to the rotation matrix passed in. Definition at line 96 of file LocationalAgent.cpp.

References LocationalAgent::mOrientation.

```
97    {
98       mOrientation = orient;
99    }
```

6.6.2.17 void setPosition (const gmtl::Point3f & *p*)

Sets the agent's current position to the value passed in. Should only be used initially for simulation setup. Definition at line 76 of file LocationalAgent.cpp.

References LocationalAgent::mPosition.

```
77    {
78       mPosition = pos;
79    }
```

6.6.2.18 void setSpeed (float *speed*) [protected]

Sets the Agent's current speed to the value passed in. Definition at line 106 of file LocationalAgent.cpp.

References LocationalAgent::mSpeed.

```
107    {
108       mSpeed = s;
109    }
```

6.6.2.19 void think (float *dt*) [virtual]

Used to solve the current problem and determine where to move

Reimplemented from **Agent** (p. 20).

Definition at line 51 of file LocationalAgent.cpp.

Referenced by MotionFlokk::think().

```
52    {
53       Agent::think( dt );
54    }
```

6.6.3 Member Data Documentation

6.6.3.1 gmtl::AABoxf mAABox [protected]

Definition at line 185 of file LocationalAgent.h.

Referenced by LocationalAgent::getAABox(), and LocationalAgent::setAABox().

6.6.3.2 gmtl::Matrix33f mDesiredOrientation [protected]

Definition at line 183 of file LocationalAgent.h.

Referenced by LocationalAgent::getDesiredOrientation(), and LocationalAgent::setDesiredOrientation().

6.6.3.3 gmtl::Point3f mDesiredPosition [protected]

Definition at line 173 of file LocationalAgent.h.

Referenced by LocationalAgent::getDesiredPosition(), and LocationalAgent::setDesiredPosition().

6.6.3.4 float mDesiredSpeed [protected]

Definition at line 179 of file LocationalAgent.h.

Referenced by LocationalAgent::getDesiredSpeed(), LocationalAgent::LocationalAgent(), and LocationalAgent::setDesiredSpeed().

6.6.3.5 gmtl::Vec3f mDirection [protected]

Definition at line 176 of file LocationalAgent.h.

Referenced by LocationalAgent::getDirection(), LocationalAgent::LocationalAgent(), and LocationalAgent::setDirection().

6.6.3.6 float mMaxSpeed [protected]

Definition at line 180 of file LocationalAgent.h.

Referenced by LocationalAgent::getMaxSpeed(), LocationalAgent::LocationalAgent(), and LocationalAgent::setMaxSpeed().

6.6.3.7 gmtl::Matrix33f mOrientation [protected]

Definition at line 182 of file LocationalAgent.h.

Referenced by LocationalAgent::getOrientation(), and LocationalAgent::setOrientation().

6.6.3.8 gmtl::Point3f mPosition [protected]

Definition at line 172 of file LocationalAgent.h.

Referenced by LocationalAgent::getPosition(), and LocationalAgent::setPosition().

6.6.3.9 float mSpeed [protected]

Definition at line 178 of file LocationalAgent.h.

Referenced by LocationalAgent::getSpeed(), LocationalAgent::LocationalAgent(), and LocationalAgent::setSpeed().

The documentation for this class was generated from the following files:

- **LocationalAgent.h**
- **LocationalAgent.cpp**

6.7 LocationalAgent::Movement Struct Reference

`#include <LocationalAgent.h>`

Public Attributes

- **LocationalAgent** * **owner**
- gmtl::Point3f **currentPos**
- gmtl::Point3f **desiredPos**
- float **currentSpeed**
- float **desiredSpeed**
- float **maxSpeed**
- void * **userData**

6.7.1 Detailed Description

Movement(p. 40) This struct includes all data needed to define an agent's desired movement for one frame. This gets passed to the **MotionSolver**(p. 44) to deal with.

Definition at line 59 of file LocationalAgent.h.

6.7.2 Member Data Documentation

6.7.2.1 gmtl::Point3f currentPos

Definition at line 62 of file LocationalAgent.h.

Referenced by MotionFlokk::think().

6.7.2.2 float currentSpeed

Definition at line 64 of file LocationalAgent.h.

Referenced by MotionFlokk::think().

6.7.2.3 gmtl::Point3f desiredPos

Definition at line 63 of file LocationalAgent.h.

Referenced by MotionFlokk::think().

6.7.2.4 float desiredSpeed

Definition at line 65 of file LocationalAgent.h.

Referenced by MotionFlokk::think().

6.7.2.5 float maxSpeed

Definition at line 66 of file LocationalAgent.h.

Referenced by MotionFlokk::think().

6.7.2.6 LocationalAgent* owner

Definition at line 61 of file LocationalAgent.h.

Referenced by MotionFlokk::think().

6.7.2.7 void* userData

Definition at line 67 of file LocationalAgent.h.

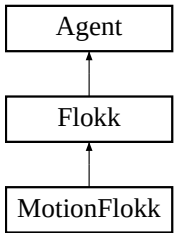
The documentation for this struct was generated from the following file:

- **LocationalAgent.h**

6.8 MotionFlokk Class Reference

`#include <MotionFlokk.h>`

Inheritance diagram for MotionFlokk::



Public Member Functions

- **MotionFlokk** ()
- virtual ~**MotionFlokk** ()
- virtual void **addAgent** (**LocationalAgent** *a)
- **MotionSolver** * **getMotionSolver** ()
- void **setMotionSolver** (**MotionSolver** *solver)
- virtual void **think** (float dt)

6.8.1 Constructor & Destructor Documentation

6.8.1.1 MotionFlokk ()

Definition at line 38 of file MotionFlokk.cpp.

```
38                              : Flokk()
39    {
40    }
```

6.8.1.2 ~MotionFlokk () [virtual]

Definition at line 42 of file MotionFlokk.cpp.

```
43    {
44    }
```

6.8.2 Member Function Documentation

6.8.2.1 void addAgent (LocationalAgent * *a*) [virtual]

register an agent with this flokk (only accepts type **LocationalAgent**(p. 33)). Definition at line 46 of file MotionFlokk.cpp.

```
47    {
49       Flokk::addAgent( a );
50    }
```

6.8.2.2 MotionSolver * getMotionSolver ()

Definition at line 72 of file MotionFlokk.cpp.

```
73    {
74       return mMotionSolver;
75    }
```

6.8.2.3 void setMotionSolver (MotionSolver * *solver*)

Definition at line 77 of file MotionFlokk.cpp.

```
78    {
79       mMotionSolver = solver;
80    }
```

6.8.2.4 void think (float *dt*) [virtual]

make every agent think and pass their desired movements to the motion solver

Reimplemented from **Agent** (p. 20).

Definition at line 52 of file MotionFlokk.cpp.

References LocationalAgent::Movement::currentPos, LocationalAgent::Movement::currentSpeed, LocationalAgent::Movement::desiredPos, LocationalAgent::Movement::desiredSpeed, LocationalAgent::getDesiredPosition(), LocationalAgent::getDesiredSpeed(), LocationalAgent::getMaxSpeed(), LocationalAgent::getPosition(), LocationalAgent::getSpeed(), LocationalAgent::Movement::maxSpeed, LocationalAgent::Movement::owner, MotionSolver::registerMovement(), MotionSolver::solve(), and LocationalAgent::think().

```
53    {
55       for ( size_t i = 0; i<mAgents.size(); ++i )
56       {
57          LocationalAgent *la = (LocationalAgent *)(mAgents[i]);
58          la->think( dt );
59          LocationalAgent::Movement m;
60          m.owner = la;
61          m.currentPos = la->getPosition();
62          m.desiredPos = la->getDesiredPosition();
63          m.currentSpeed = la->getSpeed();
64          m.desiredSpeed = la->getDesiredSpeed();
65          m.maxSpeed = la->getMaxSpeed();
66          mMotionSolver->registerMovement( m );
67       }
68
69       mMotionSolver->solve( dt );
70    }
```

The documentation for this class was generated from the following files:

- **MotionFlokk.h**
- **MotionFlokk.cpp**

6.9 MotionSolver Class Reference

`#include <MotionSolver.h>`

Public Member Functions

- **MotionSolver** (int xSubDivisions, int ySubDivisions, int zSubDivisions)
- virtual void **registerMovement** (const **LocationalAgent::Movement** &movement)

  allows an agent to register it's motion for this frame with the solver

- virtual void **solve** (float dt)

  find all conflicts in registered agent movements, and update agent positions accordingly

- void **setDomain** (**Domain** *d)
- **Domain** * **getDomain** () const

6.9.1 Detailed Description

MotionSolver(p. 44) This class listens for agent movements every frame and resolves conflicts that occur between agents that should collide or agents that would go out of bounds.

This class is responsible for updating each agent's position in the world every frame.

Definition at line 51 of file MotionSolver.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 MotionSolver (int *xSubDivisions*, int *ySubDivisions*, int *zSubDivisions*)

Definition at line 37 of file MotionSolver.cpp.

```
38    {
39       mXDivs = xSubDivisions;
40       mYDivs = ySubDivisions;
41       mZDivs = zSubDivisions;
42       mTreeArray.resize(mXDivs*mYDivs*mZDivs);
43       mDomain = NULL;
44    }
```

6.9.3 Member Function Documentation

6.9.3.1 Domain * getDomain () const

Definition at line 73 of file MotionSolver.cpp.

```
74    {
75       return mDomain;
76    }
```

6.9.3.2 void registerMovement (const LocationalAgent::Movement & *movement*) [virtual]

allows an agent to register it's motion for this frame with the solver

Definition at line 78 of file MotionSolver.cpp.

Referenced by MotionFlokk::think().

```
79    {
80       mMovements.push_back( movement );
81    }
```

6.9.3.3 void setDomain (Domain * *d*)

Definition at line 46 of file MotionSolver.cpp.

References Domain::getBoundingBox().

```
47    {
48       mDomain = d;
49       gmtl::AABoxf b = mDomain->getBoundingBox();
50       gmtl::Point3f min = b.getMin();
51       gmtl::Point3f max = b.getMax();
52
53       mOffSetx = min[0];
54       mOffSety = min[1];
55       mOffSetz = min[2];
56       if ( mOffSetx < 0 )
57       {
58          mOffSetx = -mOffSetx;
59       }
60       if ( mOffSety < 0 )
61       {
62          mOffSety = -mOffSety;
63       }
64       if ( mOffSetz < 0 )
65       {
66          mOffSetz = -mOffSetz;
67       }
68       mXPerDiv = ((max[0] + mOffSetx) - (min[0] + mOffSetx)) / mXDivs;
69       mYPerDiv = ((max[1] + mOffSety) - (min[1] + mOffSety)) / mYDivs;
70       mZPerDiv = ((max[2] + mOffSetz) - (min[2] + mOffSetz)) / mZDivs;
71    }
```

6.9.3.4 void solve (float *dt*) [virtual]

find all conflicts in registered agent movements, and update agent positions accordingly

Definition at line 83 of file MotionSolver.cpp.

References MotionSolver::Conflict::a, and MotionSolver::Conflict::b.

Referenced by MotionFlokk::think().

```
84    {
85       int spanX = 0;
86       int spanY = 0;
87       int spanZ = 0;
88
```

```
89       int * finalPositions = new int[mMovements.size()*6];
90
91
92       for ( unsigned int i = 0; i<mMovements.size(); ++i )
93       {
94          gmtl::Vec3f d = mMovements[i].desiredPos - mMovements[i].currentPos;
95          gmtl::LineSegf line;
96          line.mOrigin = mMovements[i].currentPos;
97          line.mDir = d;
98
99                  // update the agent's direction
100                 mMovements[i].owner->setDirection( line.mDir );
101
102          gmtl::Point3f dest = mMovements[i].desiredPos;
103
104                  /*
105          if ( mDomain && mDomain->clipToDomain( line, line ) )
106          {
107             dest = line.mOrigin + line.mDir;
108             //clip the desired position
109             mMovements[i].owner->setDesiredPosition( dest );
110          }*/
111
112
113          gmtl::normalize( d );
114          float speed = mMovements[i].desiredSpeed;
115          if ( mMovements[i].maxSpeed >= 0 && speed > mMovements[i].maxSpeed )
116          {
117             speed = mMovements[i].maxSpeed;
118          }
119
120          d = d * (speed * dt);
121
122
123          mMovements[i].owner->setSpeed( speed );
124
125          //if our speed is going to take us beyond the desired position, clamp
126          if ( gmtl::lengthSquared( d ) > gmtl::lengthSquared( line.mDir ) )
127          {
128             mMovements[i].owner->setPosition( dest );
129          }
130          else
131          {
132             mMovements[i].owner->setPosition( mMovements[i].currentPos + d );
133          }
134          continue;
135
136
137          //TODO: collision detection and handling with other agents
138          //TODO: orientation lerping
139
140
141          gmtl::Point3f min;
142          gmtl::Point3f max;
143
144          gmtl::Point3f currMin = (mMovements[i].owner->getAABox()).getMin();
145          gmtl::Point3f destMin = currMin + d;
146
147          gmtl::Point3f currMax = (mMovements[i].owner->getAABox()).getMax();
148          gmtl::Point3f destMax = currMax + d;
149
150          for ( unsigned int j=0; j<3; j++ )
151          {
152             if ( currMin[i]<=destMin[i] ) min[i] = currMin[i];
153             else min[i] = destMin[i];
154
155             if ( currMax[i]<=destMax[i] ) max[i] = currMax[i];
```

```
156            else max[i] = destMax[i];
157         }
158
159         //PSYCHO "Dan Shipton" ALGORITHM START
160         int xMinPos = 0;
161         int yMinPos = 0;
162         int zMinPos = 0;
163         int xMaxPos = 0;
164         int yMaxPos = 0;
165         int zMaxPos = 0;
166
167         // Find min pos to place
168         if ( !(min[0]+mOffSetx <=0) )
169         {
170            xMinPos = (int)floor(min[0] + mOffSetx / mXDivs*mXPerDiv);
171            if ( xMinPos > mXDivs )
172            {
173               xMinPos = mXDivs;
174            }
175         }
176         if ( !(min[1]+ mOffSety <=0) )
177         {
178            yMinPos = (int)floor(min[1] + mOffSety / mYDivs*mYPerDiv);
179            if ( yMinPos > mYDivs )
180            {
181               yMinPos = mYDivs;
182            }
183         }
184         if ( !(min[2]+mOffSetz <=0) )
185         {
186            zMinPos = (int)floor(min[2] + mOffSetz / mZDivs*mZPerDiv);
187            if ( zMinPos > mZDivs )
188            {
189               zMinPos = mZDivs;
190            }
191         }
192
193         if ( !(max[0]+mOffSetx <=0) )
194         {
195            xMaxPos = (int)floor(max[0] + mOffSetx / mXDivs*mXPerDiv);
196            if ( xMaxPos > mXDivs )
197            {
198               xMaxPos = mXDivs;
199            }
200         }
201         if ( !(max[1]+ mOffSety <=0) )
202         {
203            yMaxPos = (int)floor(max[1] + mOffSety / mYDivs*mYPerDiv);
204            if ( yMaxPos > mYDivs )
205            {
206               yMaxPos = mYDivs;
207            }
208         }
209         if ( !(max[2]+mOffSetz <=0) )
210         {
211            zMaxPos = (int)floor(max[2] + mOffSetz / mZDivs*mZPerDiv);
212            if ( zMaxPos > mZDivs )
213            {
214               zMaxPos = mZDivs;
215            }
216         }
217
218         if ( xMinPos != xMaxPos )
219         {
220            spanX++;
221         }
222         if ( yMinPos != yMaxPos )
```

```
223         {
224            spanY++;
225         }
226         if ( zMinPos != zMaxPos )
227         {
228            spanZ++;
229         }
230
231         finalPositions[i*6] = xMinPos;
232         finalPositions[i*6 + 1] = yMinPos;
233         finalPositions[i*6 + 2] = zMinPos;
234         finalPositions[i*6 + 3] = xMaxPos;
235         finalPositions[i*6 + 4] = yMaxPos;
236         finalPositions[i*6 + 5] = zMaxPos;
237
238
239
240      }
241
242      mMovements.clear();
243      delete [] finalPositions;
244
245               return;
246
247      //Organize tree by the least spanned variables
248      int xShift = 0;
249      int yShift = 0;
250      int zShift = 0;
251      if ( spanX<=spanY )
252      {
253         if ( spanX<=spanZ )
254         {
255            if ( spanY<=spanZ )
256            {
257               xShift = mYDivs * mZDivs;
258               yShift = mZDivs;
259               //xyz
260            }
261            else
262            {
263               xShift = mYDivs * mZDivs;
264               zShift = mYDivs;
265               //xzy
266            }
267         }
268         else
269         {
270            zShift = mYDivs * mXDivs;
271            xShift = mYDivs;
272            //zxy
273         }
274      }
275      else
276      {
277         if ( spanX <= spanZ )
278         {
279            yShift = mXDivs * mZDivs;
280            xShift = mZDivs;
281            //yxz
282         }
283         else
284         {
285            if ( spanY <= spanZ )
286            {
287               yShift = mZDivs * mXDivs;
288               zShift = mXDivs;
289               //yzx
```

```
290            }
291            else
292            {
293               zShift = mYDivs * mXDivs;
294               yShift = mXDivs;
295               //zyx
296            }
297         }
298      }
299
300
301
302      /*
303      finalPositions[i*6] = XMinPos;
304      finalPositions[i*6 + 1] = YMinPos;
305      finalPositions[i*6 + 2] = ZMinPos;
306      finalPositions[i*6 + 3] = XMaxPos;
307      finalPositions[i*6 + 4] = YMaxPos;
308      finalPositions[i*6 + 5] = ZMaxPos;
309      */
310
311      // Instert the two markers into the array that is a tree
312      for ( unsigned int i = 0; i<mMovements.size(); ++i )
313      {
314         mTreeArray[((finalPositions[i*6])*(xShift))+((finalPositions[i*6 + 1])*(yShift))+((finalPositions[i*6 + 2])
315         mTreeArray[((finalPositions[i*6 + 3])*(xShift))+((finalPositions[i*6 + 4])*(yShift))+((finalPositions[i*6 +
316      }
317
318
319
320      std::vector<MotionSolver::Conflict> conflicts;
321      std::vector<int> found;
322      // search through list and find any possible conflicts
323      for ( int i = mTreeArray.size()-1; i >= 0; i-- )
324      {
325
326         while ( !(mTreeArray[i].empty()) )
327         {
328            int search_val = mTreeArray[i].back();
329            mTreeArray[i].pop_back();
330            //search down tree for matching value and delete it adding conflicts on the way
331            //check to see if matching search val is in that space!
332            //  for every value that you see that is not it add it to possible conflicts
333            //  move down the array and keep searching
334            for(unsigned int j = 0; j < mTreeArray[i].size(); j++ )
335            {
336               if(mTreeArray[i][j] == search_val)  // found it right away
337               {
338                  std::vector<int>::iterator itr;
339                  itr+=j;
340                  mTreeArray[i].erase(itr);
341                  for(unsigned int k =0; k < mTreeArray[i].size(); k++)
342                  {
343                     // add in possible conflicts
344                     Conflict c;
345                     c.a = search_val;
346                     c.b = mTreeArray[i][k];
347                     conflicts.push_back(c);
348                  }
349                  break;
350               }
351               else
352               {
353                     Conflict c;
354                     c.a = search_val;
355                     c.b = mTreeArray[i][j];
356                     conflicts.push_back(c);
```

```
357                }
358             }
359          }
360       }
361           //Have our list of conflicts that need to be tested......
362
363
364       mMovements.clear();
365           delete [] finalPositions;
366    }
```

The documentation for this class was generated from the following files:

- **MotionSolver.h**
- **MotionSolver.cpp**

6.10 MotionSolver::Conflict Struct Reference

`#include <MotionSolver.h>`

Public Attributes

- int **a**
- int **b**

6.10.1 Member Data Documentation

6.10.1.1 int a

Definition at line 68 of file MotionSolver.h.

Referenced by MotionSolver::solve().

6.10.1.2 int b

Definition at line 69 of file MotionSolver.h.

Referenced by MotionSolver::solve().

The documentation for this struct was generated from the following file:

- **MotionSolver.h**

6.11 Problem Class Reference

`#include <Problem.h>`

Public Member Functions

- **Problem** ()
- ~**Problem** ()
- virtual void **solve** (float dt)=0
- virtual void **reset** ()=0
- virtual bool **isSolved** ()=0

Protected Member Functions

- void **setAgent** (**Agent** *targetAgent)

Protected Attributes

- **Agent** * **mAgent**

6.11.1 Constructor & Destructor Documentation

6.11.1.1 Problem ()

Definition at line 37 of file Problem.cpp.

```
38    {
39    }
```

6.11.1.2 ~Problem ()

Definition at line 41 of file Problem.cpp.

```
42    {
43    }
```

6.11.2 Member Function Documentation

6.11.2.1 virtual bool isSolved () [pure virtual]

Referenced by Agent::think().

6.11.2.2 virtual void reset () [pure virtual]

Referenced by Agent::think().

6.11.2.3 void setAgent (Agent * *targetAgent*) [protected]

Definition at line 45 of file Problem.cpp.

References Problem::mAgent.

Referenced by Agent::addProblem(), and Agent::setDefaultProblem().

```
46    {
47       mAgent = targetAgent;
48    }
```

6.11.2.4 virtual void solve (float *dt*) [pure virtual]

Referenced by Agent::think().

6.11.3 Member Data Documentation

6.11.3.1 Agent* mAgent [protected]

Definition at line 55 of file Problem.h.

Referenced by Problem::setAgent().

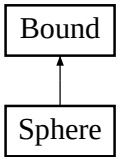
The documentation for this class was generated from the following files:

- **Problem.h**
- **Problem.cpp**

6.12 Sphere Class Reference

`#include <Sphere.h>`

Inheritance diagram for Sphere::



Public Member Functions

- **Sphere** ()
- **Sphere** (float radius)
- void **setRadius** (float radius)
- float **getRadius** () const
- virtual bool **contains** (const gmtl::Point3f &p)
- virtual bool **intersect** (const gmtl::LineSegf &line, int &numHits, float &hit1, float &hit2)
- virtual gmtl::AABoxf **getBoundingBox** () const

6.12.1 Constructor & Destructor Documentation

6.12.1.1 Sphere ()

Create a sphere of radius 1 at the origin Definition at line 37 of file Sphere.cpp.

```
38    {
39       mSphere.setCenter( gmtl::Point3f(0,0,0) );
40       mSphere.setRadius(1);
41    }
```

6.12.1.2 Sphere (float *radius*)

Create a sphere of radius 'radius' at the origin Definition at line 43 of file Sphere.cpp.

```
44    {
45       mSphere.setCenter( gmtl::Point3f(0,0,0) );
46       mSphere.setRadius( radius );
47    }
```

6.12.2 Member Function Documentation

6.12.2.1 bool contains (const gmtl::Point3f & *p*) [virtual]

Does this bound contain the point p?

Implements **Bound** (p. 22).

Definition at line 59 of file Sphere.cpp.

References Bound::getPosition(), and gmtl::intersect().

```
60    {
61       //p transformed into local space
62       gmtl::Point3f pt = gmtl::Point3f(0,0,0) + ( p - getPosition() );
63       return gmtl::intersect( mSphere, pt );
64    }
```

6.12.2.2 virtual gmtl::AABoxf getBoundingBox () const [inline, virtual]

Returns the axis-aligned bounding box of this bound

Implements **Bound** (p. 22).

Definition at line 71 of file Sphere.h.

```
72       {
73          gmtl::Point3f p = mSphere.getCenter();
74          gmtl::Point3f mn( p[0]-mSphere.getRadius(), p[1]-mSphere.getRadius(), p[2]-mSphere.getRadius() );
75          gmtl::Point3f mx( p[0]+mSphere.getRadius(), p[1]+mSphere.getRadius(), p[2]+mSphere.getRadius() );
76          return gmtl::AABoxf( mn, mx );
77       }
```

6.12.2.3 float getRadius () const

Returns the radius of the sphere. Definition at line 54 of file Sphere.cpp.

```
55    {
56       return mSphere.getRadius();
57    }
```

6.12.2.4 bool intersect (const gmtl::LineSegf & *line*, int & *numHits*, float & *hit1*, float & *hit2*) [virtual]

Determines if a given line segment intersects this bound, how many times it intersects, and where along the line segment parametrically, indicated by hit1 and hit2 which both range from 0 - 1.

Implements **Bound** (p. 23).

Definition at line 66 of file Sphere.cpp.

References Bound::getInvTransform().

```
67    {
68       gmtl::LineSegf lt = line;
69       lt.mOrigin = getInvTransform() * lt.mOrigin;
70       lt.mDir = getInvTransform() * lt.mDir;
71       return gmtl::intersectVolume( mSphere, lt, numHits, hit1, hit2 );
72    }
```

6.12.2.5 void setRadius (float *radius*)

Redefine the radius of the sphere Definition at line 49 of file Sphere.cpp.

```
50    {
51       mSphere.setRadius( radius );
52    }
```

The documentation for this class was generated from the following files:

- **Sphere.h**
- **Sphere.cpp**

Chapter 7

FLokk of Kurious Kreatures File Documentation

7.1 Agent.cpp File Reference

`#include <flokk/Agent.h>`

Namespaces

- namespace **flokk**

7.2 Agent.h File Reference

```
#include <flokk/Problem.h>
#include <vector>
#include <gmtl/gmtl.h>
```

Namespaces

- namespace **flokk**

Classes

- class **Agent**

7.3 Bound.cpp File Reference

```
#include <flokk/Bound.h>
```

Namespaces

- namespace **flokk**

7.4 Bound.h File Reference

```
#include <gmtl/gmtl.h>
```

Namespaces

- namespace **flokk**

Classes

- class **Bound**

7.5 Box.cpp File Reference

```
#include <flokk/Box.h>
#include <flokk/IntersectionExt.h>
```

Namespaces

- namespace **flokk**

7.6 Box.h File Reference

```
#include <flokk/Bound.h>
```

Namespaces

- namespace **flokk**

Classes

- class **Box**

7.7 Domain.cpp File Reference

```
#include <flokk/Domain.h>
```

Namespaces

- namespace **flokk**

7.8 Domain.h File Reference

```
#include <gmtl/gmtl.h>
#include <flokk/Bound.h>
#include <flokk/Sphere.h>
#include <flokk/Box.h>
```

Namespaces

- namespace **flokk**

Classes

- class **Domain**

7.9 Flokk.cpp File Reference

```
#include <flokk/Flokk.h>
```

Namespaces

- namespace **flokk**

7.10 Flokk.h File Reference

```
#include <flokk/Agent.h>
#include <vector>
```

Namespaces

- namespace **flokk**

Classes

- class **Flokk**

7.11 IntersectionExt.h File Reference

```
#include <gmtl/Util/Assert.h>
#include <gmtl/Vec.h>
#include <gmtl/VecOps.h>
#include <gmtl/Math.h>
#include <gmtl/Intersection.h>
```

Namespaces

- namespace **gmtl**

Defines

- #define **X** 0
- #define **Y** 1
- #define **Z** 2
- #define **CROSS**(dest, v1, v2)
- #define **DOT**(v1, v2) (v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2])
- #define **SUB**(dest, v1, v2)
- #define **FINDMINMAX**(x0, x1, x2, min, max)
- #define **AXISTEST_X01**(a, b, fa, fb)
- #define **AXISTEST_X2**(a, b, fa, fb)
- #define **AXISTEST_Y02**(a, b, fa, fb)
- #define **AXISTEST_Y1**(a, b, fa, fb)
- #define **AXISTEST_Z12**(a, b, fa, fb)
- #define **AXISTEST_Z0**(a, b, fa, fb)

Functions

- template<class T> int **planeBoxOverlap** (const Vec< T, 3 > &normal, T d, const Point< T, 3 > &maxbox)
- template<class T> int **triBoxOverlap** (const Point< T, 3 > &boxcenter, const Vec< T, 3 > &boxhalfsize, const Tri< T > &triverts)
- template<class T> bool **intersectLineAAPlane** (unsigned int axis, T offset, const Ray< T > &ray, T &t)
- template<class T> bool **intersect** (const AABox< T > &box, const Tri< T > &tri)
- template<class T> bool **intersect** (const LineSeg< T > &seg, const AABox< T > &box, T &tin, T &tout)
- template<class T> bool **intersect** (const Ray< T > &seg, const AABox< T > &box, T &tin, T &tout)

7.11.1 Define Documentation

7.11.1.1 #define AXISTEST_X01(a, b, fa, fb)

Value:

```
p0 = a*v0[Y] - b*v0[Z];                           \
                p2 = a*v2[Y] - b*v2[Z];                           \
                        if(p0<p2) {min=p0; max=p2;} else {min=p2; max=p0;} \
                rad = fa * boxhalfsize[Y] + fb * boxhalfsize[Z];   \
                if(min>rad || max<-rad) return 0;
```

Definition at line 311 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.2 #define AXISTEST_X2(a, b, fa, fb)

Value:

```
p0 = a*v0[Y] - b*v0[Z];                           \
                p1 = a*v1[Y] - b*v1[Z];                           \
                        if(p0<p1) {min=p0; max=p1;} else {min=p1; max=p0;} \
                rad = fa * boxhalfsize[Y] + fb * boxhalfsize[Z];   \
                if(min>rad || max<-rad) return 0;
```

Definition at line 318 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.3 #define AXISTEST_Y02(a, b, fa, fb)

Value:

```
p0 = -a*v0[X] + b*v0[Z];                          \
                p2 = -a*v2[X] + b*v2[Z];                          \
                        if(p0<p2) {min=p0; max=p2;} else {min=p2; max=p0;} \
                rad = fa * boxhalfsize[X] + fb * boxhalfsize[Z];   \
                if(min>rad || max<-rad) return 0;
```

Definition at line 327 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.4 #define AXISTEST_Y1(a, b, fa, fb)

Value:

```
p0 = -a*v0[X] + b*v0[Z];                          \
                p1 = -a*v1[X] + b*v1[Z];                          \
                        if(p0<p1) {min=p0; max=p1;} else {min=p1; max=p0;} \
                rad = fa * boxhalfsize[X] + fb * boxhalfsize[Z];   \
                if(min>rad || max<-rad) return 0;
```

Definition at line 334 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.5 #define AXISTEST_Z0(a, b, fa, fb)

Value:

```
p0 = a*v0[X] - b*v0[Y];                             \
                p1 = a*v1[X] - b*v1[Y];                             \
                        if(p0<p1) {min=p0; max=p1;} else {min=p1; max=p0;} \
                rad = fa * boxhalfsize[X] + fb * boxhalfsize[Y];   \
                if(min>rad || max<-rad) return 0;
```

Definition at line 350 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.6 #define AXISTEST_Z12(a, b, fa, fb)

Value:

```
p1 = a*v1[X] - b*v1[Y];                             \
                p2 = a*v2[X] - b*v2[Y];                             \
                        if(p2<p1) {min=p2; max=p1;} else {min=p1; max=p2;} \
                rad = fa * boxhalfsize[X] + fb * boxhalfsize[Y];   \
                if(min>rad || max<-rad) return 0;
```

Definition at line 343 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.7 #define CROSS(dest, v1, v2)

Value:

```
dest[0]=v1[1]*v2[2]-v1[2]*v2[1]; \
          dest[1]=v1[2]*v2[0]-v1[0]*v2[2]; \
          dest[2]=v1[0]*v2[1]-v1[1]*v2[0];
```

Definition at line 265 of file IntersectionExt.h.

7.11.1.8 #define DOT(v1, v2) (v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2])

Definition at line 270 of file IntersectionExt.h.

Referenced by gmtl::planeBoxOverlap().

7.11.1.9 #define FINDMINMAX(x0, x1, x2, min, max)

Value:

```
min = max = x0;   \
  if(x1<min) min=x1;\
  if(x1>max) max=x1;\
  if(x2<min) min=x2;\
  if(x2>max) max=x2;
```

Definition at line 277 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.10 #define SUB(dest, v1, v2)

Value:

```
dest[0]=v1[0]-v2[0]; \
          dest[1]=v1[1]-v2[1]; \
          dest[2]=v1[2]-v2[2];
```

Definition at line 272 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.11 #define X 0

Definition at line 260 of file IntersectionExt.h.

Referenced by gmtl::planeBoxOverlap(), and gmtl::triBoxOverlap().

7.11.1.12 #define Y 1

Definition at line 261 of file IntersectionExt.h.

Referenced by gmtl::triBoxOverlap().

7.11.1.13 #define Z 2

Definition at line 262 of file IntersectionExt.h.

Referenced by gmtl::planeBoxOverlap(), and gmtl::triBoxOverlap().

7.11.2 Function Documentation

7.11.2.1 bool intersect (const Ray< T > & *seg*, const AABox< T > & *box*, T & *tin*, T & *tout*)

Definition at line 169 of file IntersectionExt.h.

References gmtl::intersectLineAAPlane().

```
170    {
171       float t;
172       tin = -1;
173       tout = 10000;
174
175
176       //if parallel to the box in any way, check bounds
177       if ( seg.mDir[0] == 0 )
178       {
179          if ( seg.mOrigin[0]>box.mMax[0] || seg.mOrigin[0]<box.mMin[0] ) return false;
180       }
181       else if ( seg.mDir[1] == 0 )
182       {
183          if ( seg.mOrigin[1]>box.mMax[1] || seg.mOrigin[1]<box.mMin[1] ) return false;
184       }
185       else if ( seg.mDir[2] == 0 )
186       {
187          if ( seg.mOrigin[2]>box.mMax[2] || seg.mOrigin[2]<box.mMin[2] ) return false;
```

```
188       }
189       //test each plane of the box, and if an intersection occurs, cut away at tin and tout, until the [tin,tout] s
190       //is fully bound by the box, or tout crosses tin, in which case it was a miss.
191       //Source: "Computer Graphics using OpenGL, Second Edition" pg. 754
192
193
194       if ( intersectLineAAPlane( 0,box.mMin[0],seg,t ) )
195       {
196          if ( seg.mDir[0]>0 ) tin = t > tin ? t : tin;
197          else           tout = t < tout ? t : tout;
198
199          if ( tin>tout )return false;
200
201       }
202
203       if ( intersectLineAAPlane( 0,box.mMax[0],seg,t ) )
204       {
205          if ( seg.mDir[0]<0 ) tin = t > tin ? t : tin;
206          else           tout = t < tout ? t : tout;
207
208          if ( tin>tout )return false;
209
210       }
211
212       if ( intersectLineAAPlane( 1,box.mMin[1],seg,t ) )
213       {
214          if ( seg.mDir[1]>0 ) tin = t > tin ? t : tin;
215          else           tout = t < tout ? t : tout;
216
217          if ( tin>tout )return false;
218
219       }
220
221       if ( intersectLineAAPlane( 1,box.mMax[1],seg,t ) )
222       {
223          if ( seg.mDir[1]<0 ) tin = t > tin ? t : tin;
224          else           tout = t < tout ? t : tout;
225
226          if ( tin>tout )return false;
227
228       }
229
230       if ( intersectLineAAPlane( 2,box.mMin[2],seg,t ) )
231       {
232          if ( seg.mDir[2]>0 ) tin = t > tin ? t : tin;
233          else           tout = t < tout ? t : tout;
234
235          if ( tin>tout )return false;
236
237       }
238
239       if ( intersectLineAAPlane( 2,box.mMax[2],seg,t ) )
240       {
241          if ( seg.mDir[2]<0 ) tin = t > tin ? t : tin;
242          else           tout = t < tout ? t : tout;
243
244          if ( tin>tout )return false;
245
246       }
247
248       if ( tin>tout )return false;
249       return true;
250
251    }
```

7.11.2.2 bool intersect (const LineSeg< T > & *seg*, const AABox< T > & *box*, T & *tin*, T & *tout*)

given a line segment and an axis aligned bounding box, returns whether the line intersects the box, and if so, tin and tout are set to the parametric terms on the line segment where the segment enters and exits the box respectively Definition at line 81 of file IntersectionExt.h.

References gmtl::intersectLineAAPlane().

```
82    {
83       float t;
84       tin = -1;
85       tout = 10000;
86
87
88       //if parallel to the box in any way, check bounds
89       if ( seg.mDir[0] == 0 )
90       {
91          if ( seg.mOrigin[0]>box.mMax[0] || seg.mOrigin[0]<box.mMin[0] ) return false;
92       }
93       else if ( seg.mDir[1] == 0 )
94       {
95          if ( seg.mOrigin[1]>box.mMax[1] || seg.mOrigin[1]<box.mMin[1] ) return false;
96       }
97       else if ( seg.mDir[2] == 0 )
98       {
99          if ( seg.mOrigin[2]>box.mMax[2] || seg.mOrigin[2]<box.mMin[2] ) return false;
100       }
101       //test each plane of the box, and if an intersection occurs, cut away at tin and tout, until the [tin,tout] s
102       //is fully bound by the box, or tout crosses tin, in which case it was a miss.
103       //Source: "Computer Graphics using OpenGL, Second Edition" pg. 754
104
105
106       if ( intersectLineAAPlane( 0,box.mMin[0],seg,t ) )
107       {
108          if ( seg.mDir[0]>0 ) tin = t > tin ? t : tin;
109          else           tout = t < tout ? t : tout;
110
111          if ( tin>tout )return false;
112
113       }
114
115       if ( intersectLineAAPlane( 0,box.mMax[0],seg,t ) )
116       {
117          if ( seg.mDir[0]<0 ) tin = t > tin ? t : tin;
118          else           tout = t < tout ? t : tout;
119
120          if ( tin>tout )return false;
121
122       }
123
124       if ( intersectLineAAPlane( 1,box.mMin[1],seg,t ) )
125       {
126          if ( seg.mDir[1]>0 ) tin = t > tin ? t : tin;
127          else           tout = t < tout ? t : tout;
128
129          if ( tin>tout )return false;
130
131       }
132
133       if ( intersectLineAAPlane( 1,box.mMax[1],seg,t ) )
134       {
135          if ( seg.mDir[1]<0 ) tin = t > tin ? t : tin;
136          else           tout = t < tout ? t : tout;
137
138          if ( tin>tout )return false;
```

```
139
140       }
141
142       if ( intersectLineAAPlane( 2,box.mMin[2],seg,t ) )
143       {
144          if ( seg.mDir[2]>0 ) tin = t > tin ? t : tin;
145          else           tout = t < tout ? t : tout;
146
147          if ( tin>tout )return false;
148
149       }
150
151       if ( intersectLineAAPlane( 2,box.mMax[2],seg,t ) )
152       {
153          if ( seg.mDir[2]<0 ) tin = t > tin ? t : tin;
154          else           tout = t < tout ? t : tout;
155
156          if ( tin>tout )return false;
157
158       }
159
160       if ( tin>tout )return false;
161       tout = tout > 1 ? 1 : tout;
162       if ( tin>1 )return false;
163       return true;
164
165    }
```

7.11.2.3 bool intersect (const AABox< T > & *box*, const Tri< T > & *tri*)

given an axis aligned bounding box and a triangle, returns whether the triangle intersects the box anywhere Definition at line 69 of file IntersectionExt.h.

Referenced by Sphere::contains(), Box::contains(), and Box::intersect().

```
70    {
71       Point<T,3> center = (box.mMax + box.mMin)/2;
72       Vec<T,3> half = box.mMax - center;
73       return gmtl::triBoxOverlap<T>(center, half, tri );
74    }
```

7.11.2.4 bool intersectLineAAPlane (unsigned int *axis*, T *offset*, const Ray< T > & *ray*, T & *t*)

given an axis of 0 - 2 (for x y z) and an offset on the specified axis, calculates the intersection between the line (as defined by a ray along it) and an axis aligned plane setting t to be the parametric term on the line where it intersects the plane Definition at line 56 of file IntersectionExt.h.

Referenced by gmtl::intersect().

```
57    {
58       gmtlASSERT( axis<3 && "axis out of bounds in boolIntersectRayAAPlane(...)" );
59       if ( ray.mDir[axis] == 0 ) return false;
60       t = ( offset - ray.mOrigin[axis]) / (ray.mDir[axis]);
61       return true;
62    }
```

7.11.2.5 int planeBoxOverlap (const Vec< T, 3 > & *normal*, T *d*, const Point< T, 3 > & *maxbox*)

Definition at line 284 of file IntersectionExt.h.

References DOT, X, and Z.

```
285   {
286      int q;
287      T vmin[3],vmax[3];
288      for ( q=X;q<=Z;q++ )
289      {
290         if ( normal[q]>0.0f )
291         {
292            vmin[q]=-maxbox[q];
293            vmax[q]=maxbox[q];
294         }
295         else
296         {
297            vmin[q]=maxbox[q];
298            vmax[q]=-maxbox[q];
299         }
300      }
301      if ( DOT(normal,vmin)+d>0.0f ) return 0;
302      if ( DOT(normal,vmax)+d>=0.0f ) return 1;
303
304      return 0;
305   }
```

7.11.2.6 int triBoxOverlap (const Point< T, 3 > & *boxcenter*, const Vec< T, 3 > & *boxhalfsize*, const Tri< T > & *triverts*)

Definition at line 358 of file IntersectionExt.h.

References AXISTEST_X01, AXISTEST_X2, AXISTEST_Y02, AXISTEST_Y1, AXISTEST_Z0, AXISTEST_Z12, FINDMINMAX, SUB, X, Y, and Z.

```
359   {
360
361      /*    use separating axis theorem to test overlap between triangle and box */
362      /*    need to test for overlap in these directions: */
363      /*    1) the {x,y,z}-directions (actually, since we use the AABB of the triangle */
364      /*       we do not even need to test these) */
365      /*    2) normal of the triangle */
366      /*    3) crossproduct(edge from tri, {x,y,z}-directin) */
367      /*       this gives 3x3=9 more tests */
368      Vec<T,3> v0,v1,v2;
369      Vec<T,3> axis;
370      T min,max,d,p0,p1,p2,rad,fex,fey,fez;
371      Vec<T,3> normal,e0,e1,e2;
372
373      /* This is the fastest branch on Sun */
374      /* move everything so that the boxcenter is in (0,0,0) */
375      SUB(v0,triverts[0],boxcenter);
376      SUB(v1,triverts[1],boxcenter);
377      SUB(v2,triverts[2],boxcenter);
378
379      /* compute triangle edges */
380      SUB(e0,v1,v0);      /* tri edge 0 */
381      SUB(e1,v2,v1);      /* tri edge 1 */
382      SUB(e2,v0,v2);      /* tri edge 2 */
383
384      /* Bullet 3:  */
```

```
385       /*  test the 9 tests first (this was faster) */
386       fex = fabs(e0[X]);
387       fey = fabs(e0[Y]);
388       fez = fabs(e0[Z]);
389       AXISTEST_X01(e0[Z], e0[Y], fez, fey);
390       AXISTEST_Y02(e0[Z], e0[X], fez, fex);
391       AXISTEST_Z12(e0[Y], e0[X], fey, fex);
392
393       fex = fabs(e1[X]);
394       fey = fabs(e1[Y]);
395       fez = fabs(e1[Z]);
396       AXISTEST_X01(e1[Z], e1[Y], fez, fey);
397       AXISTEST_Y02(e1[Z], e1[X], fez, fex);
398       AXISTEST_Z0(e1[Y], e1[X], fey, fex);
399
400       fex = fabs(e2[X]);
401       fey = fabs(e2[Y]);
402       fez = fabs(e2[Z]);
403       AXISTEST_X2(e2[Z], e2[Y], fez, fey);
404       AXISTEST_Y1(e2[Z], e2[X], fez, fex);
405       AXISTEST_Z12(e2[Y], e2[X], fey, fex);
406
407       /* Bullet 1: */
408       /*  first test overlap in the {x,y,z}-directions */
409       /*  find min, max of the triangle each direction, and test for overlap in */
410       /*  that direction -- this is equivalent to testing a minimal AABB around */
411       /*  the triangle against the AABB */
412
413       /* test in X-direction */
414       FINDMINMAX(v0[X],v1[X],v2[X],min,max);
415       if ( min>boxhalfsize[X] || max<-boxhalfsize[X] ) return 0;
416
417       /* test in Y-direction */
418       FINDMINMAX(v0[Y],v1[Y],v2[Y],min,max);
419       if ( min>boxhalfsize[Y] || max<-boxhalfsize[Y] ) return 0;
420
421       /* test in Z-direction */
422       FINDMINMAX(v0[Z],v1[Z],v2[Z],min,max);
423       if ( min>boxhalfsize[Z] || max<-boxhalfsize[Z] ) return 0;
424
425       /* Bullet 2: */
426       /*  test if the box intersects the plane of the triangle */
427       /*  compute plane equation of triangle: normal*x+d=0 */
428       gmtl::cross(normal,e0,e1);
429       d=-gmtl::dot(normal,v0);  /* plane eq: normal.x+d=0 */
430       if ( !planeBoxOverlap<T>(normal,d,boxhalfsize) ) return 0;
431
432       return 1;   /* box and triangle overlaps */
433    }
```

7.12 LocationalAgent.cpp File Reference

```
#include <flokk/LocationalAgent.h>
#include <flokk/Problem.h>
#include <flokk/Flokk.h>
```

Namespaces

- namespace **flokk**

7.13 LocationalAgent.h File Reference

```
#include <flokk/Agent.h>
#include <vector>
#include <gmtl/gmtl.h>
```

Namespaces

- namespace **flokk**

Classes

- class **LocationalAgent**
- struct **LocationalAgent::Movement**

7.14 MotionFlokk.cpp File Reference

```
#include <flokk/MotionFlokk.h>
#include <flokk/MotionSolver.h>
```

Namespaces

- namespace **flokk**

7.15 MotionFlokk.h File Reference

```
#include <flokk/Flokk.h>
#include <flokk/LocationalAgent.h>
#include <vector>
```

Namespaces

- namespace **flokk**

Classes

- class **MotionFlokk**

7.16 MotionSolver.cpp File Reference

```
#include <flokk/MotionSolver.h>
#include <math.h>
```

Namespaces

- namespace **flokk**

7.17 MotionSolver.h File Reference

```
#include <vector>
#include <gmtl/gmtl.h>
#include <flokk/LocationalAgent.h>
#include <flokk/Domain.h>
```

Namespaces

- namespace **flokk**

Classes

- class **MotionSolver**
- struct **MotionSolver::Conflict**

7.18 Problem.cpp File Reference

```
#include <flokk/Problem.h>
```

Namespaces

- namespace **flokk**

7.19 Problem.h File Reference

Namespaces

- namespace **flokk**

Classes

- class **Problem**

7.20 Sphere.cpp File Reference

```
#include <flokk/Sphere.h>
```

Namespaces

- namespace **flokk**

7.21 Sphere.h File Reference

```
#include <flokk/Bound.h>
```

Namespaces

- namespace **flokk**

Classes

- class **Sphere**

Index